Politecnico di Milano - Dipartimento di Elettronica, informazione e Bioingegneria

Prof. Mauro Negri

Fondamenti di Informatica 3 settembre 2014
III appello

Matricola/codice persona_____Cognome_____Nome_____

Durata prova: 1 ora 30 minuti

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE SU QUESTO PLICO (anche sul retro dei fogli)
Si può utilizzare la matita. Vietato l'uso di cellulari, libri, eserciziari, appunti o calcolatrici.
Non è possibile uscire dall'aula durante la prova tranne in casi eccezionali.
La copiatura accertata di parti del compito può comportare la valutazione riprovato dei compiti coinvolti

Esercizio 1 (8 punti) _____

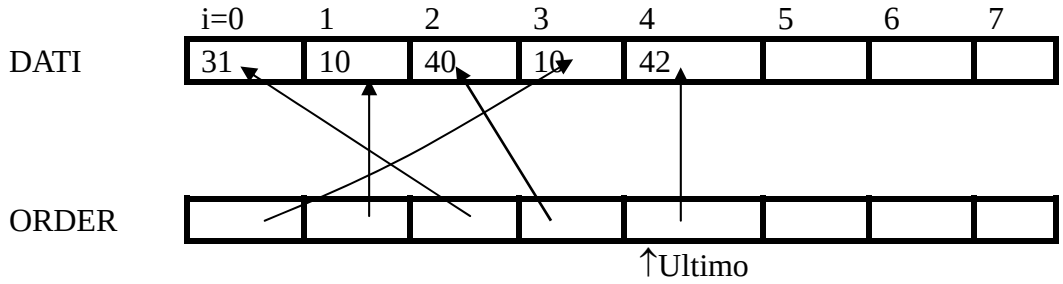
Esercizio 2 (8 punti) _____

Esercizio 3 (8 punti) _____

Esercizio 4 (6 punti) _____

Punteggio totale _____

Esercizio 1. (retro prima pagina) Siano date le seguenti definizioni globali:
#define MAX 100
int dati[MAX]; int *order[MAX]; int ultimo=-1;
dove:
- il vettore DATI contiene numeri interi anche duplicati memorizzati a partire dall'inizio senza preservare nessun ordine.
- Il vettore ORDER contiene in ogni elemento utile l'indirizzo di un elemento del vettore DATI. La scansione sequenziale del vettore ORDER, a partire dall'inizio e sino alla posizione indicata dalla variabile ultimo, permette di ottenere gli elementi di DATI in modo ordinato (per valori crescenti) come mostrato nella seguente figura:

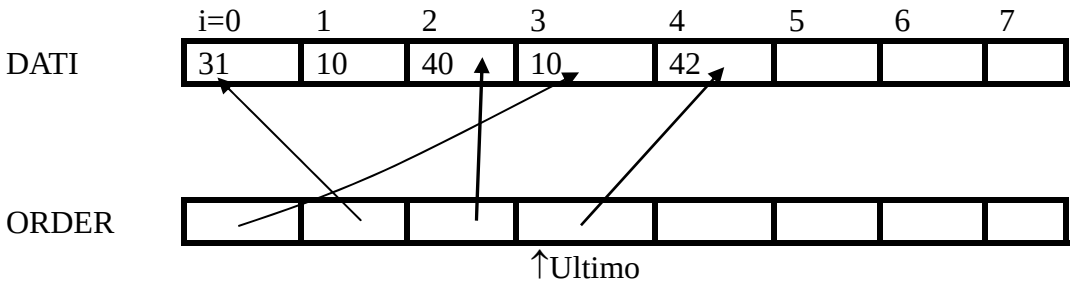


Realizzare una funzione **SCRIVI** che riceve, come parametri, il vettore ORDER e il nome di un file e provvede a scrivere nel file i valori di DATI ordinati in modo crescente, scandendo ORDER (dopo l'esecuzione della funzione il file conterrà solo i valori scritti dalla funzione – eventuale contenuto precedente è eliminato); ogni operazione di write scrive un valore di DATI sul file attraverso l'istruzione che si preferisce. La funzione ritorna, come parametro, 0 se non è possibile operare sul file (si supponga che le operazioni di scrittura/lettura e chiusura file abbiano sempre successo), 1 se il file è stato scritto e contiene almeno un numero e 2 altrimenti. La funzione DEVE lavorare solo su dati locali e parametri, pertanto passare come parametro qualsiasi ulteriore dato globale necessario.

```
Ipotesi fatta in aula
Strutture dati corrette nei puntatori, nei valori e nella variabile ultimo
(controlli non necessari nella funzione)
#include <stdlib.h>
#include <stdio.h>
#define MAX 100
int dati[MAX];  int *order[MAX];
int ultimo=-1;
struct elem { int campo1; struct elem * next; struct elem *previous;};
      struct elem *primo=NULL; //inizialmente vuota


int scrivi(int *order[], int ultimo, char *filename)
// ultimo per determinare area utile vettore order
{ FILE *fw; int i;
  if (ultimo ==-1) return 2;
  fw=fopen(filename, "wb"); if (fw==NULL) return 0;
  for (i=0; i<=ultimo; i++)
     fwrite(order[i], sizeof(int), 1, fw);
  fclose(fw);    return 1;
}
```

Esercizio 2. (retro seconda pagina) Con riferimento all'esercizio 1 si definisca una funzione **DropDuplicate** che riceve come parametri il vettore ORDER e provvede ad eliminare i duplicati nel seguente modo: la funzione scandendo ORDER quando incontra un valore duplicato in DATI lo elimina logicamente dal vettore, modificando il suo puntatore in ORDER affinché indirizzi l'elemento successivo a quello cancellato; nella seguente figura si mostrano i due vettori precedenti dopo la rimozione del duplicato 10. La funzione ritorna, come parametri, il vettore modificato e inoltre 0 se non sono stati eliminati duplicati e 1 altrimenti. La funzione DEVE lavorare solo su dati locali e parametri, pertanto passare come parametro qualsiasi ulteriore dato globale necessario.



Nel testo si capisce che se esiste un valore che appare più di una volta, se ne conserva solo il primo; gli altri vengono cancellati logicamente da dati (puntatori di order non li referenziano più).

Ipotesi fatta in aula
Strutture dati corrette nei puntatori, nei valori e nella variabile ultimo (controlli non necessari nella funzione)

Idea
Si scandisce a partire dal secondo e si verifica se è uguale al primo; in caso affermativo order slitta i puntatori di una posizione a sinistra per sovrascrivere il secondo e si ripete il controllo del nuovo secondo col primo. Altrimenti si passa a controllare il secondo col terzo ripetendo il ragionamento precedente e così via sino all'ultimo elemento indicato dalla variabile ultimo (il testo non ipotizza di usare puntatori a null per trovare l'ultimo).
Caso particolare: l'elemento da eliminare è l'ultimo.
Nota: sono possibili altri approcci.

```
int DropDuplicati(int *order[], int *ultimo)
{int eliminati=0, i, pos;
 i=1;
 while  (i<=*ultimo)
    if (*order[i]==*order[i-1])
                if (i!=*ultimo) {for(pos=i; pos<=(*ultimo-1); pos++)
                                  order[pos]=order[pos+1];
                                  (*ultimo)--; eliminati=1;
                                 }// devo riconsiderare i-esimo perché è un nuovo
                                  //valore (evitare i++)
                 else {i++;// per uscire dal ciclo
                       eliminati=1; (*ultimo)--;
   else i++;
 return eliminati;
}
```

Esercizio 3. Con riferimento all'esercizio 1 e date le seguenti definizioni globali aggiuntive:
struct elem { int campo1; struct elem * next; struct elem *previous};
struct elem *primo=NULL; //inizialmente vuota
dove i campi next e previous contengono l'indirizzo del prossimo/precedente elemento della lista e nel caso che questi non esistano conterranno NULL.

Scrivere una funzione **INS** che riceve come parametri il vettore ORDER, un intero POS che indica una posizione nel vettore ORDER(4 significa ORDER[4]) e la lista. La funzione crea un elemento dinamico della lista e vi carica il valore intero indirizzato dal puntatore di ORDER in posizione POS. L'elemento creato è memorizzato in ultima posizione nella lista. Infine la funzione ritorna, come parametri, la lista modificata e 1 se l'operazione ha avuto successo e 0 se l'operazione non è possibile. La funzione DEVE lavorare solo su dati locali e parametri, pertanto passare come parametro qualsiasi ulteriore dato globale necessario.

Non esistono ipotesi e quindi necessario controllo preliminare almeno del parametro pos.

```
int ins(int *order[], int pos, struct elem **primo, int ultimo)
 {struct elem *nuovo, *temp=*primo;

  if ((pos>ultimo)||(pos<0)) {printf("no pos\n");return 0; }

  nuovo= (struct elem *) malloc(sizeof(struct elem));
  if (nuovo== NULL) {printf("no memory\n");return 0; }

  nuovo->campo1=*order[pos]; nuovo->next=NULL; nuovo->previous=NULL;

  if (temp==NULL) *primo=nuovo; //lista vuota
  else
    {while (temp->next !=NULL) temp=temp->next;
     temp->next=nuovo; nuovo->previous=temp;
    }
return 1;}

Possibile main per verifiche (non richiesto)
void main()
{int res,i; struct elem *t;
dati[0]=31; dati[1]=10; dati[2]=40; dati[3]=10; dati[4]=42;
order[0]=&(dati[3]);
order[1]=&(dati[1]);
order[2]=&(dati[0]);
order[3]=&(dati[2]);
order[4]=&(dati[4]);
ultimo=4;
res=scrivi(order, ultimo, "nome.txt");
printf ("%d \n",res);
printf("drop %d\n",DropDuplicati(order, &ultimo));

res=ins(order, 3, &primo, ultimo);
}
```

Esercizio 4.

1. Indicare la rappresentazione in complemento a 2, utilizzando 10 bit, dei numeri:
   A = 88(base 10) 0001011000
   B = -79 (base 10) 0001001111 -> 1110110001

2. Data la definizione della variabile float v= - 50,777 descrivere la sua rappresentazione in virgola mobile in base allo standard IEEE754 semplice precisione FP32 descritto a lezione;
   a) Definire prima il numero nel formato normalizzato intermedio (TRONCARE parte decimale alla **seconda** cifra decimale):

$$-1{,}58 * 2^{5}$$

   b) Codificare poi il numero normalizzato usando il formato FP32:
   S (1 bit di segno): 1
   M (23 bit per la mantissa – fermarsi ai primi 6 bit): 100101
   E (8 bit per l'esponente): 10000100

   c)Indicare quale sia il vero valore (in base 10) descritto dalla sequenza di bit memorizzata in S, M ed E del punto b) precedente (si consiglia di esprimere la mantissa in termini frazionari).
   - 50,5

   d) indicare con quale regola sia codificato l'esponente E e perché (brevemente)
   regola esponente codificato= 127 + esponente potenza del 2; in questo modo l'esponente codificato non sarà mai un numero negativo

3. Tradurre il seguente programma C in assembler

```
…
int  V [10], C;
int fun(int *p1, int *p2)
    {int V[3];….}
int main ()
    { V[8]=10;
      C=fun (V, &C);
      }
```

V: .RES 10
C:.RES 1
STACK: .RES 1000
INIZIO: MOV #STACK, SP
ADD #999, SP
 	
MOV #V, R1
ADD #8, R1
MOV #10, (R1)
 	
ADD #-1,SP
MOV #V, (SP)
ADD #-1,SP
MOV #C, (SP)
ADD #-1,SP
JSR F
	ADD #3, SP
	MOV (SP), C
	EXIT
	
	F: MOV R0, (SP)
	ADD #-1,SP
	MOV SP, R0
	ADD #-3, SP
	…
	ADD #4, SP
	MOV (SP), R0
	RTS
	.END INIZIO